In [1]:

```python
# linear algebra
import numpy as np

# data processing
import pandas as pd

# data visualization
import seaborn as sns
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import style

# Algorithms
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
```

In [2]:

```python
test_df = pd.read_csv(r"D:\Downloads\test.csv")
train_df = pd.read_csv(r"D:\Downloads\train.csv")
```

In [3]:

```python
train_df
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [4]:

```
total = train_df.isnull().sum().sort_values(ascending=False)
percent_1 = train_df.isnull().sum()/train_df.isnull().count()*100
percent_2 = (round(percent_1, 1)).sort_values(ascending=False)
missing_data = pd.concat([total, percent_2], axis=1, keys=['Total', '%'])
missing_data.head(5)
```

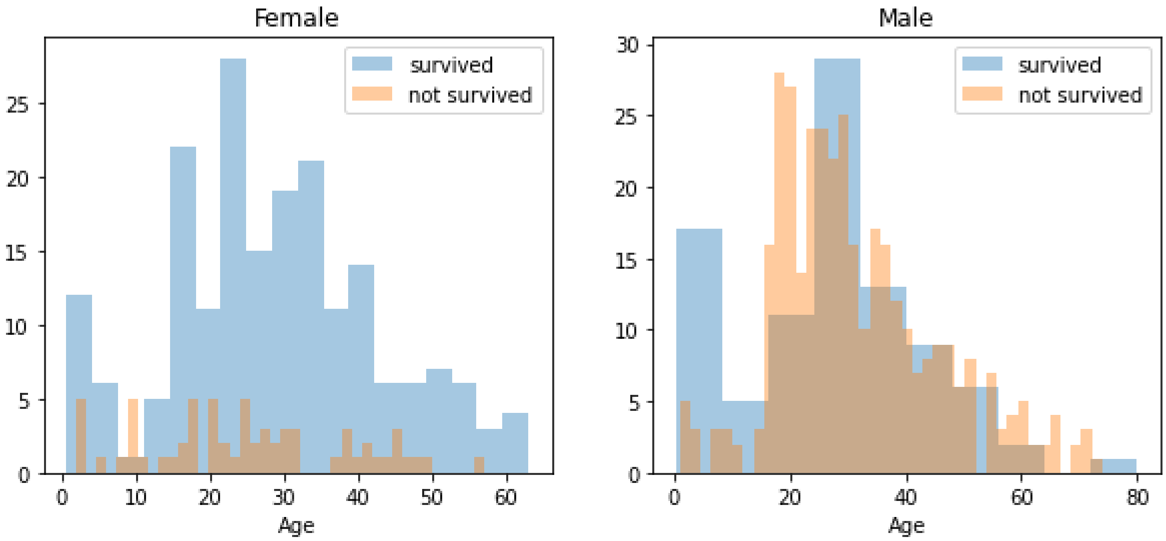
Out[4]:

	Total	%
Cabin	687	77.1
Age	177	19.9
Embarked	2	0.2
PassengerId	0	0.0
Survived	0	0.0

In [5]:

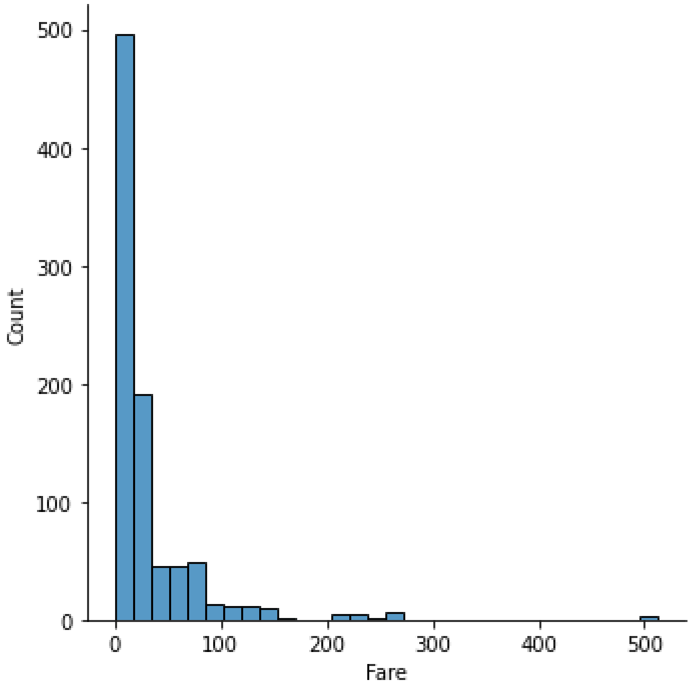
```
survived = 'survived'
not_survived = 'not survived'
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10, 4))
women = train_df[train_df['Sex']=='female']
men = train_df[train_df['Sex']=='male']
ax = sns.distplot(women[women['Survived']==1].Age.dropna(), bins=18, label = survived, a
ax = sns.distplot(women[women['Survived']==0].Age.dropna(), bins=40, label = not_survive
ax.legend()
ax.set_title('Female')
ax = sns.distplot(men[men['Survived']==1].Age.dropna(), bins=10, label = survived, ax =
ax = sns.distplot(men[men['Survived']==0].Age.dropna(), bins=40, label = not_survived, a
ax.legend()
_ = ax.set_title('Male')
```

```
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\distributions.py:26
19: FutureWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function with sim
ilar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [6]:
```
sns.displot(data=train_df, x="Fare", kde=False,bins=30)
```

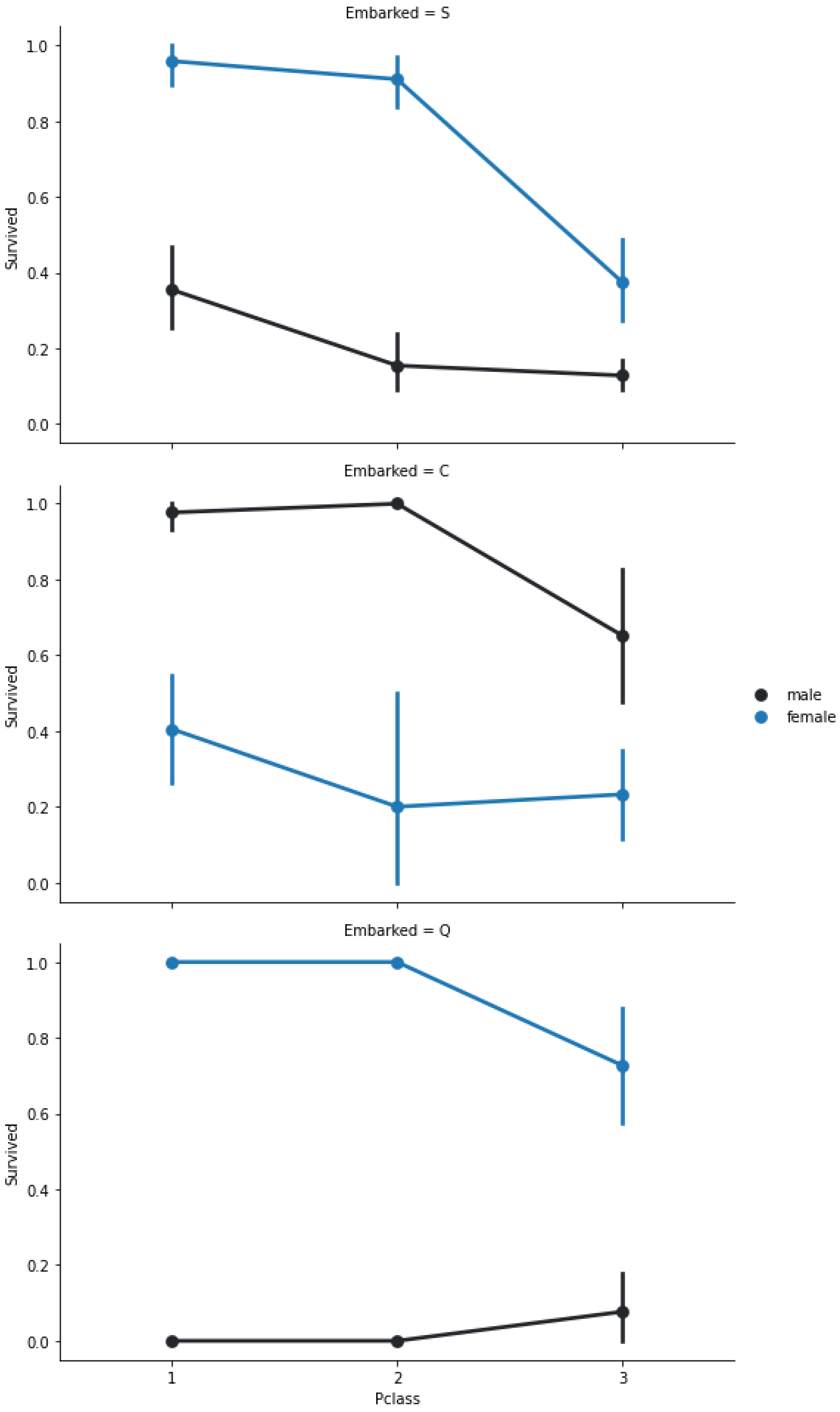
Out[6]:
```
<seaborn.axisgrid.FacetGrid at 0x25f4f68d610>
```



In [7]:
```
FacetGrid = sns.FacetGrid(train_df, row='Embarked', size=4.5, aspect=1.6)
FacetGrid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette=None,  order=None, hue
FacetGrid.add_legend()
```

```
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\axisgrid.py:337: Us
erWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[7]:
```
<seaborn.axisgrid.FacetGrid at 0x25f4f69adf0>
```

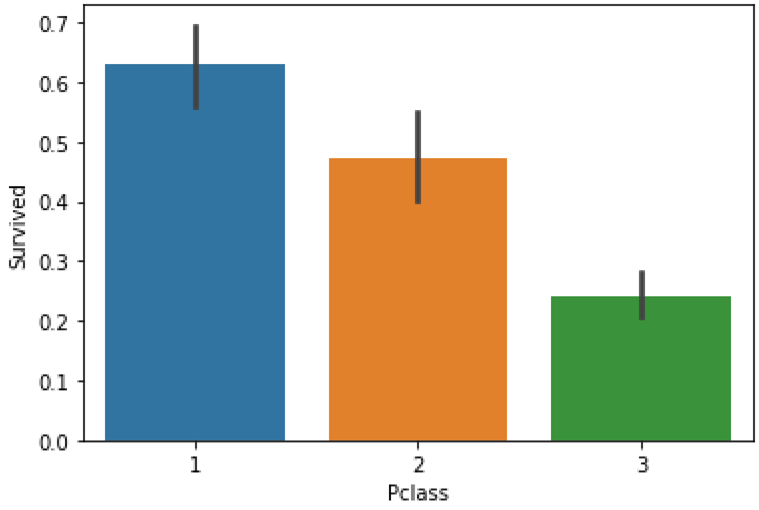


In [8]:

```
sns.barplot(x='Pclass', y='Survived', data=train_df)
```

Out[8]:

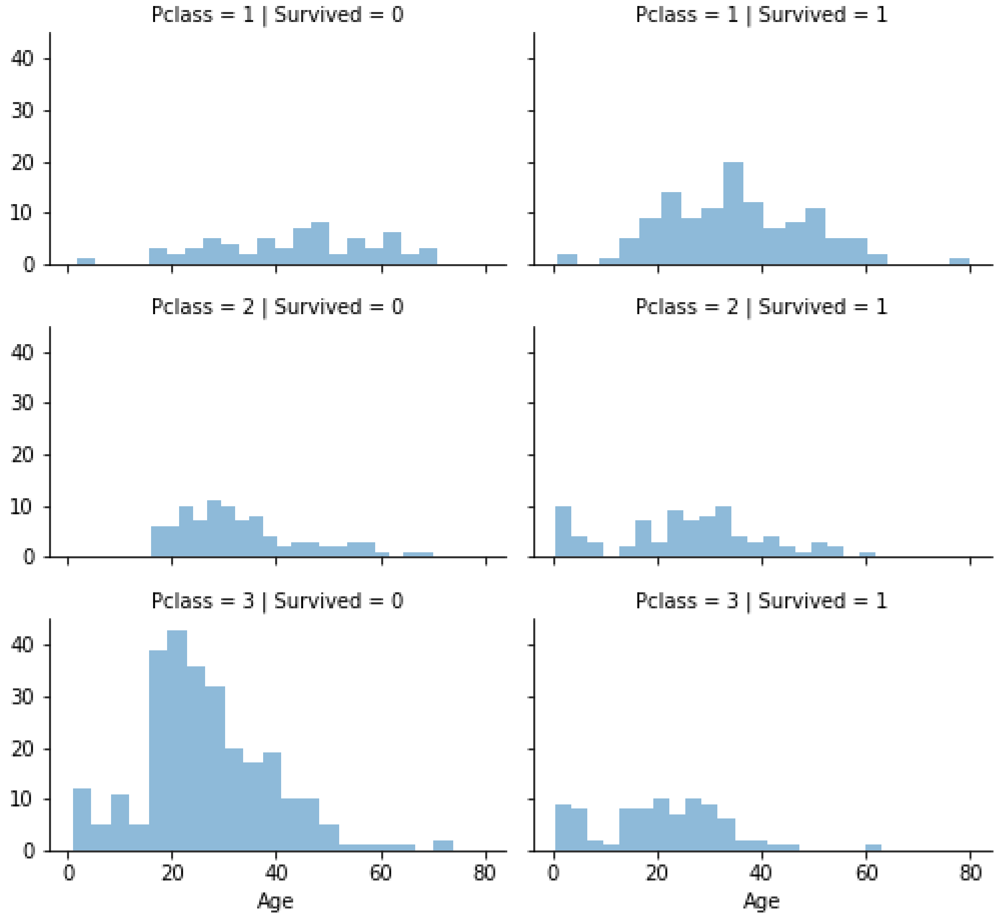
```
<AxesSubplot:xlabel='Pclass', ylabel='Survived'>
```



In [9]:

```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', size=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```

```
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\axisgrid.py:337: Us
erWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



In [10]:

```
data = [train_df, test_df]
for dataset in data:
    dataset['relatives'] = dataset['SibSp'] + dataset['Parch']
    dataset.loc[dataset['relatives'] > 0, 'not_alone'] = 0
    dataset.loc[dataset['relatives'] == 0, 'not_alone'] = 1
    dataset['not_alone'] = dataset['not_alone'].astype(int)
train_df['not_alone'].value_counts()
```

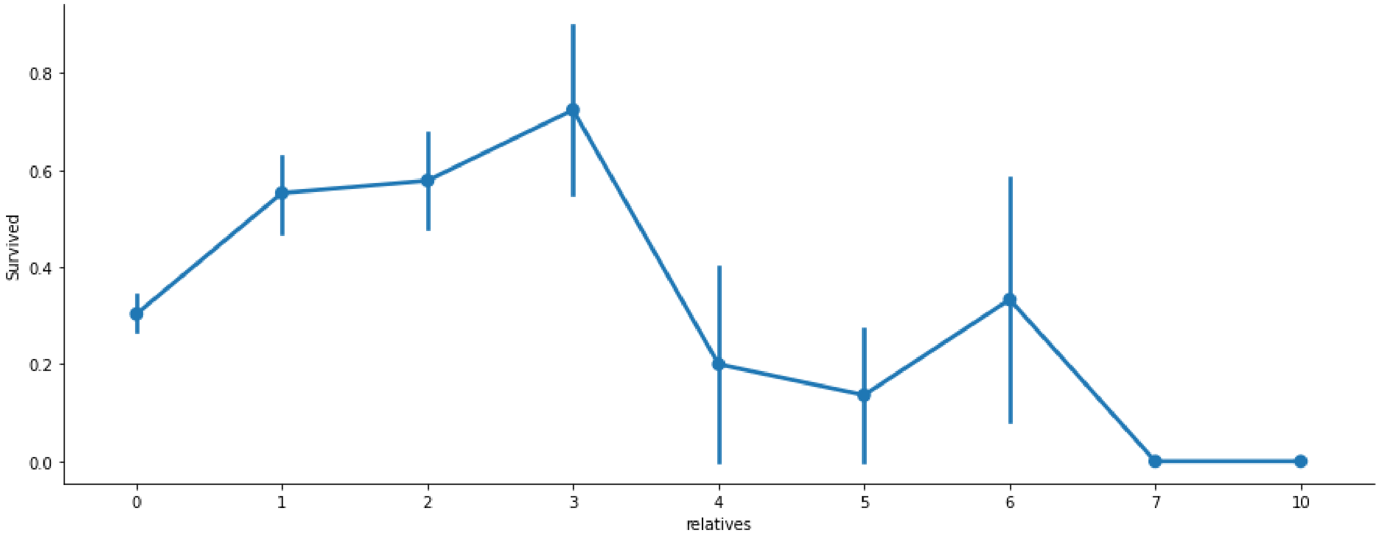
Out[10]:

```
1    537
0    354
Name: not_alone, dtype: int64
```

In [11]:

```
axes = sns.factorplot('relatives','Survived',
                      data=train_df, aspect = 2.5, )
```

```
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\categorical.py:371
7: UserWarning: The `factorplot` function has been renamed to `catplot`. The original na
me will be removed in a future release. Please update your code. Note that the default `
kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
C:\Users\rishu\AppData\Roaming\Python\Python39\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, th
e only valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [12]:

```
train_df = train_df.drop(['PassengerId','Name','Ticket'], axis=1)
```

In [13]:

```
train_df.describe()
```

Out[13]:

	Survived	Pclass	Age	SibSp	Parch	Fare	relatives	not_alone
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208	0.904602	0.602694
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429	1.613459	0.489615
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400	0.000000	0.000000
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200	0.000000	1.000000
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000	1.000000	1.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200	10.000000	1.000000

In [14]:

```
data = [train_df, test_df]

for dataset in data:
    mean = dataset["Age"].mean()
    std = dataset["Age"].std()
    is_null = dataset["Age"].isnull().sum()
    # compute random numbers between the mean, std and is_null
    rand_age = np.random.randint(mean - std, mean + std, size = is_null)
    # fill NaN values in Age column with random values generated
    age_slice = dataset["Age"].copy()
    age_slice[np.isnan(age_slice)] = rand_age
    dataset["Age"] = age_slice
    dataset["Age"] = train_df["Age"].astype(int)
train_df["Age"].isnull().sum()
```

Out[14]:

```
0
```

In [15]:

```
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [train_df, test_df]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").search(x)
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int)
# we can now drop the cabin feature
train_df = train_df.drop(['Cabin'], axis=1)
test_df = test_df.drop(['Cabin'], axis=1)
```

In [16]:

```
common_value = 'S'
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].fillna(common_value)
```

In [17]:

```
for dataset in data:
    dataset['Fare'] = dataset['Fare'].fillna(0)
    dataset['Fare'] = dataset['Fare'].astype(int)
```

In [18]:

```
genders = {"male": 0, "female": 1}
data = [train_df, test_df]

for dataset in data:
    dataset['Sex'] = dataset['Sex'].map(genders)
```

In [19]:

```
ports = {"S": 0, "C": 1, "Q": 2}
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

In [20]:

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age'] = dataset['Age'].astype(int)
    dataset.loc[ dataset['Age'] <= 11, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 11) & (dataset['Age'] <= 18), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 18) & (dataset['Age'] <= 22), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 22) & (dataset['Age'] <= 27), 'Age'] = 3
    dataset.loc[(dataset['Age'] > 27) & (dataset['Age'] <= 33), 'Age'] = 4
    dataset.loc[(dataset['Age'] > 33) & (dataset['Age'] <= 40), 'Age'] = 5
    dataset.loc[(dataset['Age'] > 40) & (dataset['Age'] <= 66), 'Age'] = 6
    dataset.loc[ dataset['Age'] > 66, 'Age'] = 6

# let's see how it's distributed
test_df['Age'].value_counts()
```

Out[20]:

```
4    79
5    73
6    68
2    61
3    61
1    43
```

```
0    33
Name: Age, dtype: int64
```

In [21]:

```
data = [train_df, test_df]
for dataset in data:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[(dataset['Fare'] > 31) & (dataset['Fare'] <= 99), 'Fare']   = 3
    dataset.loc[(dataset['Fare'] > 99) & (dataset['Fare'] <= 250), 'Fare']   = 4
    dataset.loc[ dataset['Fare'] > 250, 'Fare'] = 5
    dataset['Fare'] = dataset['Fare'].astype(int)
```

In [22]:

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age_Class']= dataset['Age']* dataset['Pclass']
```

In [23]:

```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test  = test_df.copy()
```

In [24]:

```
#X_train
```

In [25]:

```
#logreg = LogisticRegression()
#logreg.fit(X_train, Y_train)

#Y_pred = logreg.predict(X_test)

#acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
```

In [26]:

```
#X.drop(['Survived'],axis=1,inplace=True)
```

In [27]:

```
#X
```

In [28]:

```
test_df.drop(['PassengerId','Name','Ticket'],axis=1,inplace=True)
```

In [29]:

```
test_df
```

Out[29]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Age_Class
0	3	0	2	0	0	0	2	0	1	8	6
1	3	1	5	1	0	0	0	1	0	8	15
2	2	0	3	0	0	1	2	0	1	8	6
3	3	0	5	0	0	1	0	0	1	8	15
4	3	1	5	1	1	1	0	2	0	8	15
...	...	...	...	...	...	...	...	...	...	...	...
413	3	0	4	0	0	1	0	0	1	8	12
414	1	1	6	0	0	4	1	0	1	3	6
415	3	0	1	0	0	0	0	0	1	8	3
416	3	0	5	0	0	1	0	0	1	8	15
417	3	0	1	1	1	2	1	2	0	8	3

418 rows × 11 columns

In [30]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
X_train
```

Out[32]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Age_Class
0	3	0	2	1	0	0	0	1	0	8	6
1	1	1	5	1	0	3	1	1	0	3	5
2	3	1	3	0	0	0	0	0	1	8	9
3	1	1	5	1	0	3	0	1	0	3	5
4	3	0	5	0	0	1	0	0	1	8	15
...	...	...	...	...	...	...	...	...	...	...	...
886	2	0	3	0	0	1	0	0	1	8	6
887	1	1	2	0	0	2	0	0	1	2	2
888	3	1	2	1	2	2	0	3	0	8	6
889	1	0	3	0	0	2	1	0	1	3	3
890	3	0	4	0	0	0	2	0	1	8	12

891 rows × 11 columns

In [33]:
```
Y_train=train_df['Survived']
```

In [34]:
```
X_test=test_df.copy()
```

In [35]:
```
X_test
```

Out[35]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Age_Class
0	3	0	2	0	0	0	2	0	1	8	6
1	3	1	5	1	0	0	0	1	0	8	15
2	2	0	3	0	0	1	2	0	1	8	6
3	3	0	5	0	0	1	0	0	1	8	15
4	3	1	5	1	1	1	0	2	0	8	15
...	...	...	...	...	...	...	...	...	...	...	...
413	3	0	4	0	0	1	0	0	1	8	12
414	1	1	6	0	0	4	1	0	1	3	6
415	3	0	1	0	0	0	0	0	1	8	3
416	3	0	5	0	0	1	0	0	1	8	15
417	3	0	1	1	1	2	1	2	0	8	3

418 rows × 11 columns

In [36]:
```
log=LogisticRegression()
```

In [37]:
```
y_test=Y_train[:418]
```

In [38]:
```
log.fit(X_train,Y_train)
```

Out[38]:
```
LogisticRegression()
```

In [39]:
```
Y_pred=log.predict(X_test)
```

In [40]:
```
Y_pred
```

Out[40]:
```
array([0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [41]:
```
acc_log=round(log.score(X_train,Y_train)*100,2)
```

In [42]:
```
acc_log
```

Out[42]:
```
80.47
```

In [43]:
```
from sklearn.metrics import log_loss, accuracy_score,confusion_matrix
```

In [44]:
```
accuracy = accuracy_score(y_test, Y_pred)
```

In [45]:
```
accuracy
```

Out[45]:
```
0.5023923444976076
```

In [46]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [58]:
```
knn = KNeighborsClassifier(n_neighbors =37)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
```

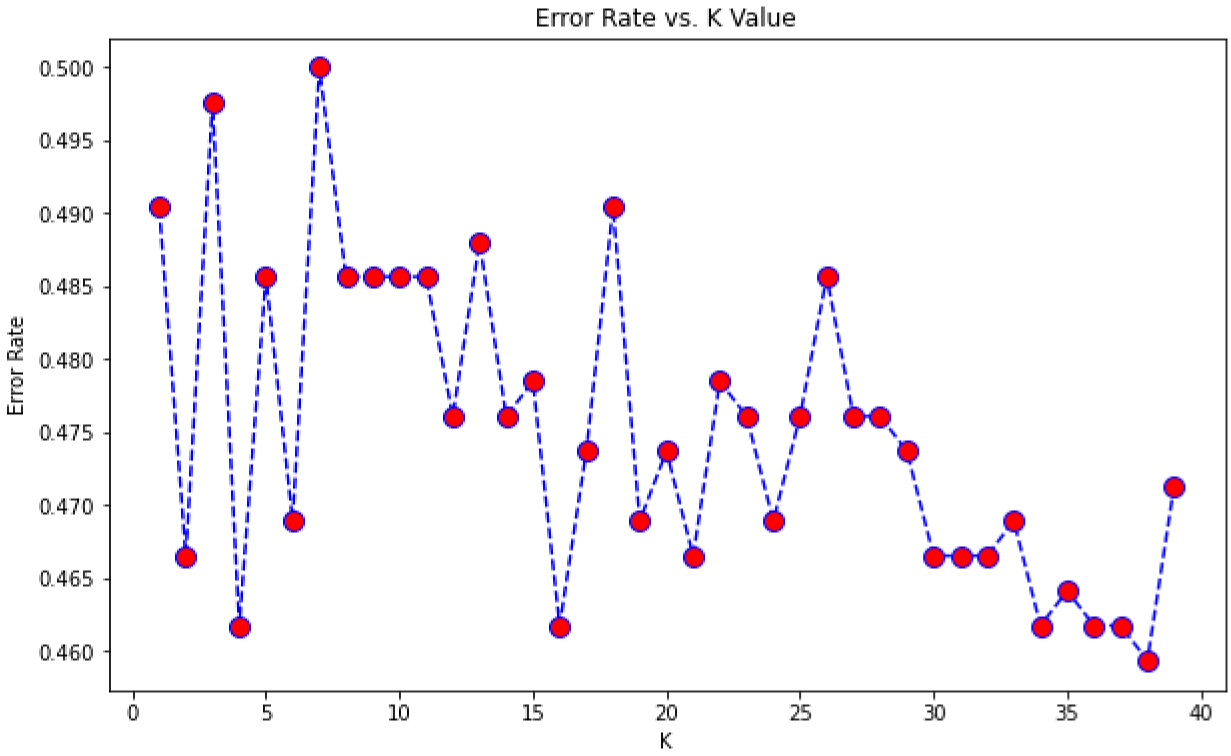
In [59]:
```
acc_knn
```

Out[59]:
```
75.65
```

In [57]:
```
error_rate = []
for i in range(1,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,Y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))
```

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K =",error_rate.index(min(error_rate)))
```

```
Minimum error:- 0.45933014354066987 at K = 37
```

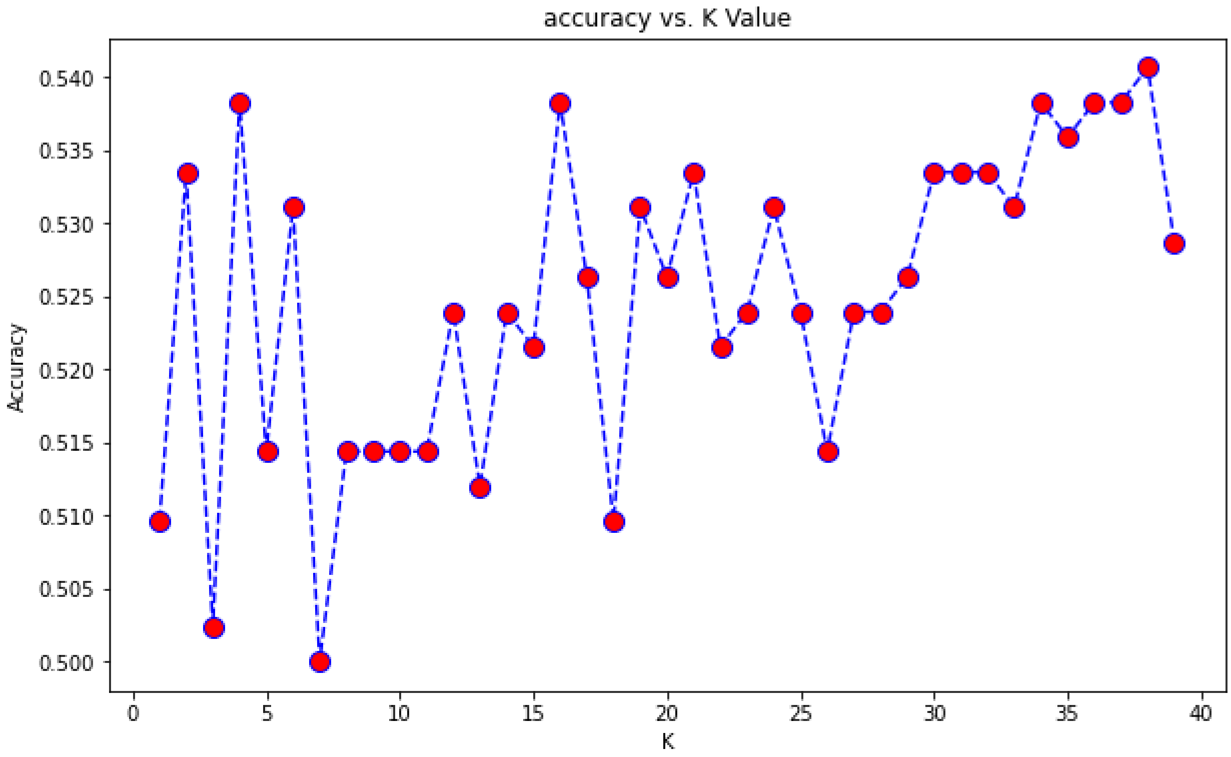


In [60]:

```
acc = []
# Will take some time
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(X_train,Y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))
```

```
Maximum accuracy:- 0.5406698564593302 at K = 37
```


accuracy vs. K Value
Accuracy
0.540
0.535
0.530
0.525
0.520
0.515
0.510
0.505
0.500
0
5
10
15
20
25
30
35
40
K

```
In [ ]:
```